## Amendments to the Claims

1. (Currently Amended) A method comprising:

receiving a system definition and a request for dependency information, the system definition comprising metadata describing a system at a level of abstraction;

requesting, via an application programming interface, a dependency collection, by sending to the application programming interface a system dependency creation request comprising the received system definition and a dependency request comprising a target logical abstraction, the target logical abstraction indicating a chosen structural level of complexity of a system described by the system definition;

receiving, responsive to the request[[s]] sent via the application programming interface, a dependency collection for the target logical abstraction comprising logical abstractions in one or more dependency chains with the target logical abstraction, wherein the dependency collection comprises logical abstractions outside of a subsystem for the target logical abstraction and the dependency chains represent chained dependencies between subsystems; and

displaying a representation of the collection.

2. (Original) The method of claim 1 wherein the system definition is received as a file identification.

3. (Original) The method of claim 1 wherein the system definition is received via a graphical user interface.

4. (Original) The method of claim 1 wherein the target logical abstraction is an unchanged logical abstraction.

5. (Original) The method of claim 1 wherein the dependency collection comprises logical abstractions dependent on the target logical abstraction.

6. (Original) The method of claim 1 wherein the dependency collection comprises logical abstractions on which the target logical abstraction depends.

7. (Original) The method of claim 1 wherein the target logical abstraction is a changed logical abstraction.

8. (Original) The method of claim 1 wherein the target logical abstraction comprises a basic block, a procedure, or a binary file.

9. (Original) The method of claim 1 wherein the dependency collection comprises a basic block logical abstraction.

10. (Original) The method of claim 1 wherein the dependency collection comprises at least one of a procedure logical abstraction or a binary file logical abstraction.

11. (Original) The method of claim 1 wherein the dependency collection comprises a named object logical abstraction or a node logical abstraction.

12. (Previously Presented) The method of claim 1, wherein the representation of the collection comprises a number of affected logical abstractions.

13. (Previously Presented) The method of claim 1, wherein the representation of the collection comprises a graphical presentation of a dependency chain.

14. (Previously Presented) The method of claim 1, wherein the representation of the collection comprises a list of logical abstractions.

15. (Currently Amended) The method of claim 1, ~~wherein the~~ further comprising displaying a representation of the collection comprises a graph of logical abstractions.

16. (Original) The method of claim 1 wherein the dependency collection further comprises logical abstractions inside the logical abstraction's subsystem.

17. (Original) The method of claim 1 wherein the logical abstraction comprises a proposed change, and a metric for indicating proposed change risk is displayed.

18. (Currently Amended) The method of claim 17 wherein the metric for proposed change risk comprises a number of logical abstractions ~~effected~~ affected by the proposed change in a relation to a total number of logical abstractions.

19. (Original) The method of claim 18 wherein the relation is further adapted with a logarithmic function.

20. (Original) A computer-readable medium having executable instructions for performing the method of claim 1.

21. (Currently Amended) A method comprising:

determining dependency information about binary files comprising entry and exit points for the binary files;

propagating dependency information about binary files to determine subsystem dependency information comprising entry and exit points for a subsystem of a system;

propagating the subsystem dependency information to determine system dependency information comprising entry and exit points for the system;

marking changed logical abstractions;

marking unchanged logical abstractions dependent on marked changed logical abstractions in other subsystems;

comparing test coverage to marked changed logical abstractions and to marked unchanged logical abstractions; and

prioritizing tests based on maximum test coverage of marked changed logical abstractions and marked unchanged logical abstractions;

performing the prioritized tests according to test priorities to produce test results.

22. (Original) The method of claim 21 wherein the test coverage comprises tests for one subsystem.

23. (Original) The method of claim 21 wherein the test coverage comprises tests for plural subsystems.

24. (Original) The method of claim 23 wherein the test coverage comprises tests for plural subsystems and maximum test coverage is considered for marked changed logical abstractions and marked unchanged logical abstractions for said plural subsystems.

25. (Currently Amended) A computer-based service comprising:

means for determining binary dependencies, comprising entry and exist points, for a defined system;

means for propagating binary dependencies to identify binaries dependent on binaries in other subsystems;

means for storing determined and propagated dependencies;

means for marking changes;

means for propagating marked changes using the determined and propagated dependencies; and

means for prioritizing tests based on test coverage of marked changes and propagated marked changes;

means for performing prioritized tests according to test priorities to produce test results.

26. (Original) The service of claim 25 wherein the means for marking changes includes means for marking proposed changes.

27. (Currently Amended) A computer-readable medium having executable instructions for performing a method comprising:

creating a system definition comprising metadata describing a system at a level of abstraction in response to receiving graphical user interface input;

receiving a dependency information request via graphical user interface input;

requesting, via an application programming interface exposed by a dependency framework, a dependency collection, by sending to the application programming interface a

system dependency creation request comprising the system definition, and a target logical abstraction identifiable from the dependency information request and indicating a chosen structural level of complexity of the system; and

receiving, responsive to the request[[s]] sent via the application programming interface, a dependency collection for the target logical abstraction comprising logical abstractions in one or more dependency chains representing chained dependencies between subsystems with the target logical abstraction;

wherein the dependency collection comprises logical abstractions outside of a subsystem for the logical abstraction.

28. (Original) The computer-readable medium of claim 27 wherein the dependency collections further comprises logical abstractions inside the target logical abstraction's subsystem.

29. (Currently Amended) A computer system comprising:

a processor coupled to volatile and nonvolatile memory;

binary files stored in memory;

software stored in memory comprising computer executable instructions for:

determining dependency information for binary files comprising entry and exit points for the binary files;

propagating dependency information about binary files to determine subsystem dependency information comprising entry and exit points for a subsystem of a system; and

propagating subsystem dependency information to determine system dependency information comprising entry and exit points for the system;

marking logical abstractions changed from a previous version;

propagating marked changes according to the dependency information comprising marking unchanged logical abstractions dependent on marked changes in other subsystems;

comparing test coverage to marked changed logical abstractions and to marked unchanged logical abstractions; and

prioritizing tests based on maximum test coverage of marked changed logical abstractions and marked unchanged logical abstractions.

30. (Original) The computer system of claim 29, wherein maximum test coverage is based on the total number of marked changed and marked unchanged logical abstractions touched by a test system wide.

31. (Original) The computer system of claim 29, wherein maximum test coverage is based on the sum of the total number of marked changed logical abstractions in a first subsystem touched by a test, and marked unchanged logical abstractions touched by the test in the first subsystem, wherein the marked unchanged logical abstractions depend on marked changed logical abstractions in other subsystems.

32. (Currently Amended) A user interface service comprising:

means for accepting a system definition comprising metadata describing a system at a level of abstraction and indicating binary files in plural subsystems;

means for accepting an indication of a target logical abstraction indicating a chosen structural level of complexity of the system; and

means for displaying dependency relationships between the target logical abstraction and a set of logical abstractions in binary files ~~from~~ between two or more of the plural subsystems.

33. (Original) The user interface service of claim 32 further comprising means for displaying a proposed change risk.

34. (Original) The user interface service of claim 32 further comprising means for displaying a change risk metric.

35. (Original) The user interface service of claim 32 further comprising means for displaying a graph of relative risk for plural subsystems.

36. (Original) The user interface service of claim 32 further comprising means for displaying test coverage evaluation results.